Pandit/Priest Booking System – Developer Project Brief

Version 1 (MVP) – Web (React) • Mobile (React Native) • Backend (ASP.NET Core) • Database (SQL Server)

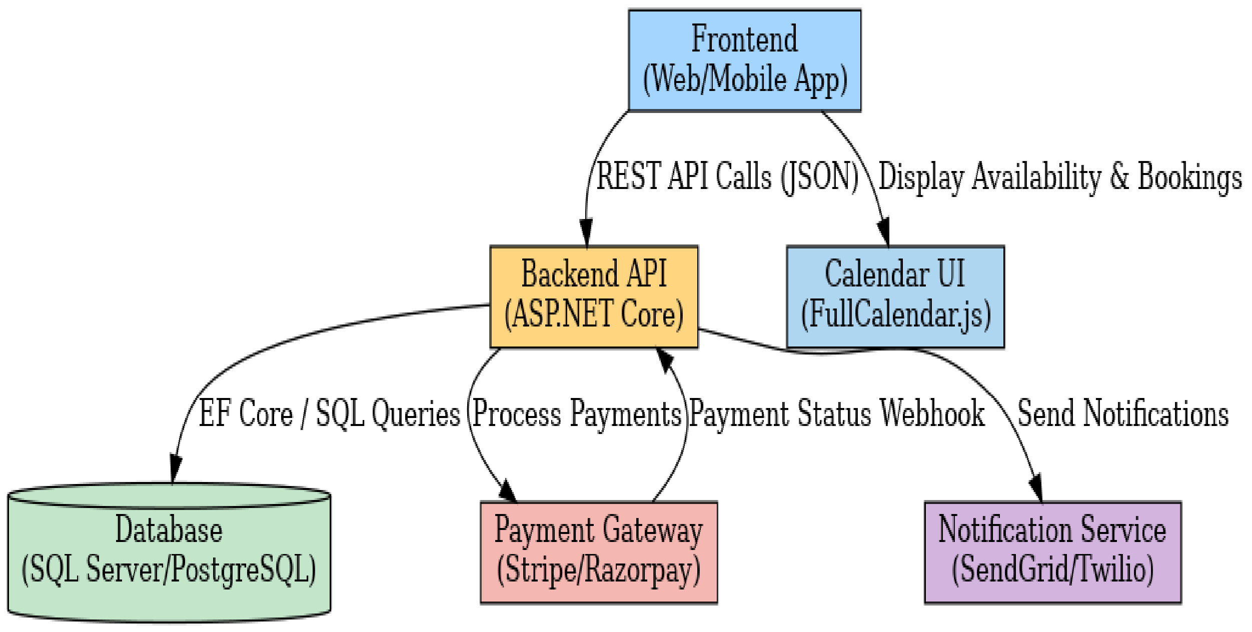
1. Overview

A multi-platform booking platform enabling users to schedule religious services with available priests. Users select a service, date/time, and optionally a specific priest. Priests can manage availability, accept bookings, and receive payments. The system supports notifications and calendar visualization for both users and priests.

2. Final Tech Stack

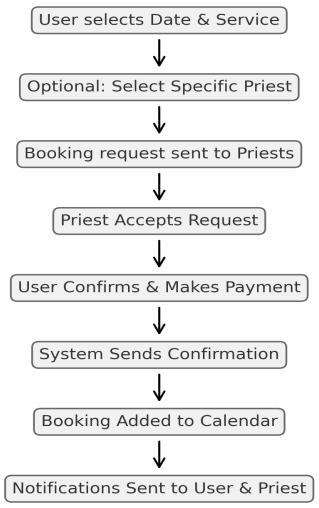
- **Frontend (Web):** React, Material UI, TailwindCSS, Redux Toolkit, React Router, FullCalendar.js, React-Toastify
- **Mobile:** React Native (Expo), React Navigation, NativeBase/React Native Paper, FCM push notifications, react-native-calendars
- **Backend:** ASP.NET Core 8 Web API (C#), REST (JSON), ASP.NET Identity + JWT auth
- **Database:** SQL Server, Entity Framework Core 8, Azure SQL Database
- **Payments:** Stripe (global) / Razorpay (India) with webhooks
- **Notifications:** SendGrid (email), Twilio (SMS), FCM (mobile push)
- **DevOps/Hosting:** Azure App Service (API), Azure Static Web Apps (Web), GitHub Actions CI/CD, Azure App Insights + Serilog

3. Architecture Diagram

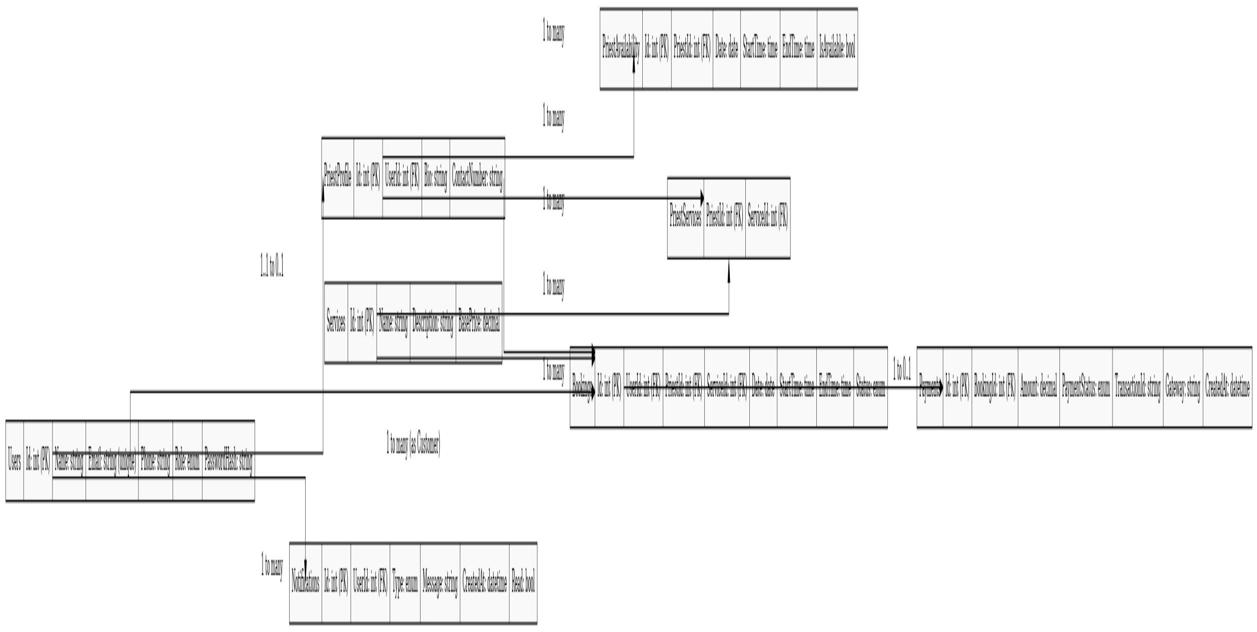


Frontend (Web/Mobile App)
REST API Calls (JSON)
Display Availability & Bookings
Backend API (ASP.NET Core)
Calendar UI (FullCalendar.js)
EF Core / SQL Queries
Process Payments
Payment Status Webhook
Send Notifications
Database (SQL Server/PostgreSQL)
Payment Gateway (Stripe/Razorpay)
Notification Service (SendGrid/Twilio)

4. Booking Workflow


User selects Date & Service
Optional: Select Specific Priest
Booking request sent to Priests
Priest Accepts Request
User Confirms & Makes Payment
System Sends Confirmation
Booking Added to Calendar
Notifications Sent to User & Priest

5. Database Schema (ERD)


PriestAvailability | Id: int (PK) | PriestId: int (FK) | Date: date | StartTime: time | EndTime: time | IsAvailable: bool
PriestProfile | Id: int (PK) | UserId: int (FK) | Bio: string | ContactNumber: string
PriestServices | PriestId: int (FK) | ServiceId: int (FK)
Services | Id: int (PK) | Name: string | Description: string | BasePrice: decimal
Bookings | Id: int (PK) | UserId: int (FK) | PriestId: int (FK) | ServiceId: int (FK) | Date: date | StartTime: time | EndTime: time | Status: enum
Payments | Id: int (PK) | BookingId: int (FK) | Amount: decimal | PaymentStatus: enum | TransactionId: string | Gateway: string | CreatedAt: datetime
Users | Id: int (PK) | Name: string | Email: string (unique) | Phone: string | Role: enum | PasswordHash: string
Notifications | Id: int (PK) | UserId: int (FK) | Type: enum | Message: string | CreatedAt: datetime | Read: bool
1 to many
1 to many
1 to many
1..1 to 0..1
1 to many
1 to many
1 to 0..1
1 to many (as Customer)
1 to many

6. Core Tables

Users(Id PK, Name, Email unique, Phone, Role enum[User,Priest,Admin], PasswordHash)
PriestProfile(Id PK, UserId FK→Users, Bio, ContactNumber)
Services(Id PK, Name, Description, BasePrice)
PriestServices(PriestId FK→PriestProfile, ServiceId FK→Services) // many-to-many
PriestAvailability(Id PK, PriestId FK→PriestProfile, Date, StartTime, EndTime, IsAvailable)
Bookings(Id PK, UserId FK→Users, PriestId FK→PriestProfile, ServiceId FK→Services, Date, StartTime, EndTime, Status enum[Pending,Accepted,Paid,Completed,Cancelled])
Payments(Id PK, BookingId FK→Bookings, Amount, PaymentStatus enum[Pending,Paid,Failed], TransactionId, Gateway, CreatedAt)
Notifications(Id PK, UserId FK→Users, Type enum[email,sms,push], Message, CreatedAt, Read)

7. API Endpoints (Draft)

- **Auth:** POST /api/auth/register, POST /api/auth/login, GET /api/auth/me
- **Users:** GET /api/users/{id}, PUT /api/users/{id}
- **Services:** GET /api/services, POST /api/services (admin), PUT /api/services/{id}
- **Priests:** GET /api/priests, GET /api/priests/{id}, PUT /api/priests/{id}, GET /api/priests/{id}/availability
- **Availability:** POST /api/availability, PUT /api/availability/{id}, GET /api/availability?priestId=&date;=
- **Bookings:** POST /api/bookings (create), GET /api/bookings/{id}, GET /api/bookings?userId=&priestId;=, PUT /api/bookings/{id}/status
- **Payments:** POST /api/payments/intent, POST /api/payments/webhook, GET /api/payments/{bookingId}
- **Notifications:** GET /api/notifications, PUT /api/notifications/{id}/read

8. Non-Functional Requirements

- **Security:** JWT auth, role-based access, parameterized queries via EF Core, HTTPS only.
- **Performance:** Index on (PriestId, Date, StartTime), pagination on list endpoints.
- **Reliability:** Payment webhook idempotency keys, retry logic for email/SMS.
- **Observability:** Structured logs (Serilog), request tracing, metrics in Azure App Insights.

9. UI Flow (Web & Mobile)

- **User:** Home → Select Service → Pick Date/Time → Choose Priest → Confirm → Pay → Success → My Bookings (Calendar)
- **Priest:** Dashboard → Manage Availability → View Requests → Accept/Decline → Confirm Booking → Calendar

10. Next Steps

- Create GitHub repo with backend (ASP.NET Core) and web frontend (React).
- Define EF Core models & run initial migration for schema above.
- Scaffold authentication (ASP.NET Identity + JWT).
- Implement core endpoints: Services, Priests, Availability, Bookings.
- Integrate FullCalendar on web with bookings API.
- Add Stripe/Razorpay sandbox and webhook handler.